# Software Design Specifications

# for

# <Hostel Management System>

Prepared by: Code Monkey's

<Author> Om Sai Vikranth
<Position> N/A

## Document Information

| | |
|---|---|
| **Title: HMS** | |
| **Project Manager: N/A** | **Document Version No: 2.0** |
| | **Document Version Date:8-04-2025** |
| **Prepared By: Code Monkeys** | **Preparation Date:5-04-2025** |

## Version History

| Ver. No. | Ver. Date | Revised By | Description | Filename |
|---|---|---|---|---|
| | | | | |
| | | | | |
| | | | | |
| | | | | |

## Table of Contents

# 1 Introduction

## 1.1 Purpose

The purpose of this document is to present a detailed software design for the Hostel Management System (HMS). It serves as a bridge between requirements and implementation, guiding developers through system architecture, component design, and data handling. The primary audience includes developers, testers, project managers, and stakeholders.

## 1.2 Scope

This SDS applies to the Hostel Management System, which manages room allocations, student records, fee payments, staff management, complaints, and other hostel operations. It affects both administrative and student users of the system.

## 1.3 Definitions, Acronyms, and Abbreviations

- HMS – Hostel Management System
- UI – User Interface
- DB – Database
- CRUD – Create, Read, Update, Delete
- API – Application Programming Interface

## 1.4 References

- Hostel Management System SRS Document
- Hostel Management System Use Case Diagrams
- UML Modelling Guidelines
- IEEE 1016-2009 – Software Design Descriptions Standard

# 2 Use Case View

This section presents key use cases that represent significant functionalities of the Hostel Management System.

2.1 Use Case

Use Case 1: Student Registration & Login

Description: Enables students to register for hostel services using their application number, email, and password.
Actors: Student, Admin

Steps:

1. Admin adds student credentials (application number, email, password) into the system.
2. Student logs in using these credentials.
3. System verifies and redirects to the student dashboard.
4. Admin can monitor registration and login logs.

Use Case 2: Hostel Admission Form Submission

Description: Allows a student to fill out a detailed hostel form including academic and personal details.
Actors: Student

Steps:

1. Student logs in and accesses the "Hostel Form" section.
2. Student fills in all required personal, academic, and emergency details.
3. System validates the input and stores it in the student forms collection.
4. Upon re-login, the same form is visible in a non-editable mode.

Use Case 3: Room Allocation

Description: Allows students to select hostel rooms, which are verified by wardens/admins.
Actors: Student, Warden, Admin

Steps:

1. Student navigates to “Apply for Room” and selects a preferred room & bed.
2. System checks form submission and room availability.
3. Room selection is stored with the student’s profile.
4. Warden/Admin can verify allocation from their dashboard.
5. System updates room status and links it to the student.

Use Case 4: Fee Payment

Description: Handles hostel fee payments made by students.
Actors: Student, Admin

Steps:

1. Student views total fee amount in the “Fees” section.
2. Payment is made through an integrated payment gateway.
3. Admin verifies the payment status.
4. System marks payment as “Paid” and issues a receipt.

Use Case 5: Complaint/Request Handling

Description: Allows students to raise issues or complaints, which are handled by admins.
Actors: Student, Admin

Steps:

1. Student navigates to the complaint/request section.
2. Submits details such as issue type, description, and room number.
3. Admin reviews, assigns, and updates resolution status.
4. Student is notified of the update.

# 3 Design Overview

The Hostel Management System follows a layered architecture, ensuring maintainability and scalability. The system consists of three primary layers:

Presentation Layer: Web-based UI for interaction.

Business Logic Layer: Core functionalities such as room allocation, fee processing.

Data Access Layer: Handles all database interactions.

## 3.1 Design Goals and Constraints

Platform-independent and accessible via browser.

Secure authentication and authorization.

Responsive design for mobile and desktop.

Efficient data storage and retrieval.

## 3.2 Design Assumptions

Users have basic digital literacy.

Reliable internet connection is available.

System will be deployed on a local server initially.

## 3.3 Significant Design Packages

User Management Package

Room Management Package

Complaint Management Package

Fee Management Package

## 3.4 Dependent External Interfaces

| External Application and Interface Name | Module Using the Interface | Functionality/ Description |
|---|---|---|
| Payment API | Payment Module | Integrates with a payment gateway (e.g., Razorpay, Paytm) to handle secure online fee transactions. |
| Mail Gateway API | Notification Module | Sends emails to students' personal email accounts (e.g., Gmail) for confirmations and alerts |

## 3.5 Implemented Application External Interfaces (and SOA web services)

The table below lists the implementation of public interfaces this design makes available for other applications.

| Interface Name | Module Implementing the Interface | Functionality/ Description |
|---|---|---|
| REST API | Backend Server | RESTful endpoints allow communication between frontend and backend.<br>Example: POST /api/complaints is used to submit a new complaint. GET /api/rooms/available returns a list of vacant rooms. |
| Admin Dashboard | Web Client | A secure web-based UI for admins to manage student records, rooms, complaints, and fees.<br>Example: Admin logs in, views a table of all complaints, clicks "Resolve" on pending ones, and downloads reports |

# 4 Logical View

## 4.1 Design Model

The Hostel Management System is structured using the 3-Tier Architecture, a modular design approach that separates the application into three distinct layers—Presentation Layer, Business Logic Layer, and Data Access Layer. This promotes better scalability, maintainability, and security.

### 1. Presentation Layer (Client Tier / Frontend)

- This is the user interface that interacts directly with students, wardens, and admins.
- Built using React.js, styled with **CSS** and responsive layouts.
- Handles input from the user and displays results from backend responses.

Examples:

- A student fills the hostel admission form.
- Admin views a dashboard with a table of students and room availability.
- Warden searches for a student to check form and room status.

### 2. Business Logic Layer (Application Tier / Server)

- Contains all core logic and validation for processing requests.
- Built using Node.js with Express.js.
- Handles routing, authorization, room allocation logic, fee status checks, etc.
- Interfaces between the frontend and the database.

Examples:

- Validates hostel form data before storing it.
- Handles room selection and prevents multiple allocations.
- Determines if a student has paid the fees before allowing room allocation.

### 3. Data Access Layer (Database Tier)

- Responsible for interacting with the MongoDB database.
- Stores all persistent data including:
    - Student records
    - Hostel form submissions
    - Room and bed information
    - Payment status
    - Leave history (for future integration)

Examples:

- Saves submitted hostel form data to the student forms collection.
- Fetches a student's form by application number for view-only display.
- Updates room allocation and payment info.

## 4.2 Use Case Realization

This section describes how the Room Allocation process works in your Hostel Management System from a student's action to system updates. It highlights the real-world interaction between users and backend logic across your 3-tier architecture.

**Actors Involved**

- Student – selects their desired room via the dashboard.
- Warden – approves and verifies allocation (future enhancement).
- System – handles request validation, room availability checks, database update, and frontend response.

**Room Allocation Flow**

1. Student logs in through the Student Dashboard using their application number, email, and password.
2. From the dashboard menu, the student clicks "Apply for Room" → "New Room".
3. The system displays a visual layout of available rooms and beds (A/B).
4. The student selects an available room and bed.
5. The frontend sends a POST /api/allocate-room request to the backend with:
    - Application number
    - Selected room number and bed
6. The backend (Node.js + Express) handles the request by:
    - Validating student's existence using the application number.
    - Checking the availability of the selected room and bed.
    - Ensuring the student has submitted their hostel form and paid fees (future enhancement).
    - Updating the room's occupancy status to "Occupied" and linking it with the student.
7. The updated information is persisted in MongoDB (rooms and student's collections).
8. A success message is sent back to the frontend
9. The frontend updates:
    - The selected bed turns red (or marked as occupied).
10. The student's dashboard shows the updated room and bed details under "My Room Details".

# 5 Data View

## 5.1 Domain Model

The domain model captures the core entities and their conceptual relationships in the Hostel Management System. These are the "real-world" objects the system deals with.

Key Entities & Descriptions
Student

Represents a user of the system who applies for a room, submits complaints, and makes payments.

Attributes: student_id, name, email, phone, room_no, etc.

Room

Represents a physical room in the hostel.

Attributes: room_no, type, status (Vacant/Occupied), capacity

Admin

System manager who allocates rooms, reviews complaints, and generates reports.

Attributes: admin_id, name, email, role

Complaint

A concern or issue raised by a student.

Attributes: complaint_id, student_id, text, status (Pending/Resolved), date_filed

Payment

Tracks hostel fee payments made by students.

Attributes: payment_id, student_id, amount, status, date, method (UPI/Card)

### 5.2 Data Model (persistent data view)

This view defines how the domain model is implemented and persisted in the database, typically visualized using an Entity-Relationship Diagram (ERD).

Key Relationships
One Student → One Room

Each student is assigned exactly one room (1:1).

Student.room_no → foreign key referencing Room.room_no.

One Admin → Many Students

An admin may manage multiple student records (1:M).

This can be tracked in logs or allocation history.

One Student → One Complaint (at a time)

While in real life students can raise many complaints, if the system tracks only active ones, then this is treated as 1:1.

### 5.2.1 Data Dictionary

This section outlines the structure and meaning of key fields used in the system's persistent data model:

student_id is an integer that uniquely identifies each student and acts as the primary key in the student database table.

room_no is a variable character field representing the room number assigned to two students. This field links student records with the room allocation.

complaint_text is a text field containing the full description of any complaint raised by a student. It supports multiline input for detailed reporting.

admin_id is an integer used to uniquely identify each administrator within the system. It helps in mapping administrative actions such as room allocations and complaint responses.

payment_id is a unique integer that represents each hostel payment transaction made by a student. It serves as a reference in the financial records.

amount is a decimal value indicating the payment amount made by the student. It reflects the transaction value in the system.

status is a variable character field used to store the current status of a payment or complaint. Common values include "Pending", "Completed", or "Resolved".

method is a variable character field denoting the mode of payment used by the student, such as "UPI", "Card", or "Cash".

## 6 Exception Handling

1. Login Failure

- Cause: Student/Admin/Warden enters incorrect application number/email or password.
- Exception Type: AuthenticationError
- System Response:
    - Message: "Invalid credentials. Please check and try again."
    - Prevents dashboard access until valid login.

2. Form Resubmission Attempt

- Cause: Student tries to submit the hostel admission form more than once.
- Exception Type: DuplicateFormSubmissionError
- System Response:
    - Message: "Form already submitted. You cannot submit again."
    - Displays previously submitted form in read-only mode.
    - No new data is written to the database.

3. Room Selection without Form Submission

- Cause: Student tries to select a room before submitting the hostel form.
- Exception Type: UnauthorizedActionError
- System Response:
    - Message: "Please submit your hostel form before selecting a room."
    - Redirects user to the Hostel Form page.

4. Room Already Occupied

- Cause: Selected room is already booked by another student.
- Exception Type: RoomConflictError
- System Response:
    - Message: "Room already occupied. Please choose another room."
    - Refreshes room availability on the interface.

5. Form Fetch Failure

- Cause: Backend fails to retrieve a student's hostel form using the application number.
- Exception Type: DataFetchError
- System Response:
    - Message: "Could not load your form. Please try again later."
    - Logged for backend investigation.

6. Server or Network Error

- Cause: API server is down or network request times out.
- Exception Type: ServerConnectionError
- System Response:
    - Message: "Something went wrong. Please check your internet connection or try again later.

# 7 Configurable Parameters

This table describes the simple configurable parameters (name/value pairs).

| Parameter Name | Description | Can Be Modified |
|---|---|---|
| Max Occupants Per Room | Maximum number of students allowed per room | No |
| Registration Open Date | Date when student registration opens | No |
| Registration Close Date | Date when student registration closes | Yes |
| Fee Due Reminder Days | Number of days before the due date to send a reminder | Yes |
| Password Min Length | Minimum number of characters for user passwords | No |
| Session Timeout | Session timeout duration (in minutes) | No |
| Admin Email | Email address for system alerts and communication | No |

# 8 Quality of Service

This section outlines how the Hostel Management System addresses quality concerns such as availability, security, performance, and operational monitoring.

## 8.1 Availability

The system is designed for high availability to ensure that hostel operations can be managed without downtime during critical hours

Design Measures to Support Availability:

downtime during maintenance by scheduling updates during holidays.

Fail-safe operations for core features like registration and room allocation — if database connectivity fails, a queue-based retry mechanism ensures data isn't lost.

Downtime Considerations:
Mass data operations (e.g., bulk room reassignments) may require brief read-only periods.

Maintenance window is scheduled for every three months (during holidays).

## 8.2 Security and Authorization

The system enforces strong security and role-based access control (RBAC) to protect sensitive user data and administrative functions.

Key Security Features:
Passwords are encrypted.

Role-Based Access:

Admin: Full access to all system operations.

Warden: Can manage rooms and view complaints.

Student: Can register, view room and fee status, and raise complaints.

User Access Management: Admins can create/edit/delete user accounts and assign roles. Students cannot alter any records

## 8.3 Load and Performance Implications

The system is optimized to handle multiple concurrent users, especially during peak registration periods.

Load Handling Strategies:
Lazy loading of non-critical data to reduce load time.

Pagination in list views (rooms, students, complaints) to avoid overloading the UI and DB.

Performance Benchmarks (Expected):
Support up to 500 concurrent users with acceptable response time.

Registration: ≤5 sec

Fee Payment: ≤ 1Min per transaction

Complaint Submission: ≤15 sec

Data Growth Projections:
Anticipated ~1500 students per semester.

Tables such as fees, complaints, and logs are expected to grow steadily—archive policy after 2 years.

### 8.4 Monitoring and Control

The system includes basic tools and hooks for runtime monitoring and error tracking.

Admin dashboard displays:

Number of active users

Room occupancy summary

Pending complaints.